Laboratorijska vežba 3

Opis standardnih kombinacionih mreža korišćenjem VHDL jezika - 2

Koder

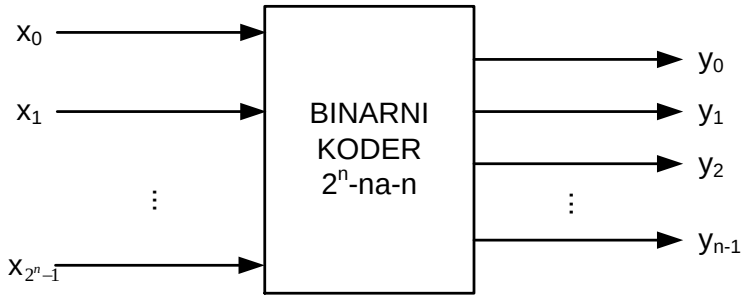
Koder je standardna kombinaciona mreža koja vrši kodovanje date informacije u neku drugu, najčešće kompaktniju formu. U praksi se najčešće koriste dve vrste kodera:

- Binarni koderi
- Prioritetni koderi
- Konvertori koda

Koderi se najčešće koriste za redukciju broja bitova potrebnih za reprezentaciju odgovarajuće informacije. Imajući ovo u vidu koderi se koriste prilikom prenosa podataka u digitalnim sistemima. Kodovanje informacije omogućava nam da transmisioni link realizujemo sa manjim brojem fizičkih linija. Kodovanje takođe može biti korisno i prilikom smeštanja podataka jer je za kodovanu verziju potreban manji broj bita za njeno čuvanje.

Binarni koder

Binarni koder koduje informaciju sa 2^n ulaznih portova u n-bitni kod. Standardni interfejs binarnog kodera prikazan je na slici 1 i sastoji se iz 2^n ulaznih portova, širine 1 bit, i n izlaznih portova, takođe širine 1 bit.



Slika 1. Interfejs generičkog binarnog kodera

U normalnom režimu rada binarnog kodera tačno jedan od ulaza trebalo bi da ima vrednost 1, a stanje na izlazima u tom slučaju predstavlja binarni broj koji identifikuje ulazni port čija je vrednost jednaka 1. Na primer, tablica istinitosnih dodela binarnog kodera 4-na-2 prikazana je dole.

x_3	x_2	x_1	x_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Primetimo da izlaz y_0 ima vrednost 1 kada su ulazi x_1 ili x_3 na 1, a izlaz y_1 ima vrednost 1 kada su ulazi x_2 ili x_3 na 1.

VHDL modeli binarnog kodera

Da bi smo ilustrovali način modelovanja koristićemo binarni koder 4-na-2. Prilikom razvoja VHDL modela prvi korak je da se deklariše odgovarajući entitet, korišćenjem VHDL *entity* deklaracije. U našem primeru binarni koder 4-na-2 ima sledeće portove:

- Četiri1-bitna ulazna porta za podatke, x_0, x_1, x_2 i x_3
- dva 1-bitna izlaza za podatke, y_0 i y_1

Odogovarajuća *entity* deklaracija binarnog kodera 4-na-2 ima sledeći izgled

```
library ieee;
use ieee.std_logic_1164.all;

entity koder4na2 is
        port (x0:       in std_logic;   -- ulazni port podataka 0
              x1:       in std_logic;   -- ulazni port podataka 1
              x2:       in std_logic;   -- ulazni port podataka 2
              x3:       in std_logic;   -- ulazni port podataka 3
              y0:       out std_logic;  -- izlazni port podataka 0
              y1:       out std_logic);-- izlazni port podataka 1
end entity koder4na2;
```

Nakon što smo opisali interfejs binarnog kodera, potrebno je modelovati njegovu funkcionalnost. Za ovo je potrebno napisati odgovarajuće arhitekturno telo. Obzirom da se binarni koder može modelovati na razne načine, možemo napisati veliki broj različitih arhitekturnih tela.

NAPOMENA: U narednim arhitekturnim telima pretpostavka je da su svi ulazni i izlazni portovi tipa *std_logic*.

Bihevijani modeli

Varijanta 1: Model binarnog kodera baziran na korišćenju konkurentne naredbe uslovne dodele vrednosti signalu

```
architecture beh1 of koder4na2 is
begin
        y0 <= '1' when x3 = '1' and x2 = '0' and
                        x1 = '0' and x0 = '0' else
              '1' when x3 = '0' and x2 = '0' and
                        x1 = '1' and x0 = '0' else
              '0';

        y1 <= '1' when x3 = '1' and x2 = '0' and
                        x1 = '0' and x0 = '0' else
              '1' when x3 = '0' and x2 = '1' and
                        x1 = '0' and x0 = '0' else
```

```
                    '0';
end architecture beh1;
```

Varijanta 2: Model binarnog kodera baziran na korišćenju *process* i *if* naredbe

```
architecture beh2 of koder4na2 is
begin
        koder: process (x0, x1, x2, x3) is
        begin
                if (x3 = '1' and x2 = '0' and
                    x1 = '0' and x0 = '0') then
                        y0 <= '1';

                elsif (x3 = '0' and x2 = '0' and
                        x1 = '1' and x0 = '0') then
                        y0 <= '1';

                else
                        y0 <= '0';
                end if;

                if (x3 = '1' and x2 = '0' and
                    x1 = '0' and x0 = '0') then
                        y1 <= '1';

                elsif (x3 = '0' and x2 = '1' and
                        x1 = '0' and x0 = '0') then
                        y1 <= '1';

                else
                        y1 <= '0';
                end if;
        end process;
end architecture beh2;
```

Kombinovanjem entity deklaracije i odgovarajućeg arhitekturnog tela dobijamo kompletan model binarnog kodera 4-na-2. Na primer, dole je prikazan model binarnog kodera 4-na-2 koji se bazira na funkcionalnom modelu koji koristi konkurentnu naredbu uslovne dodele vrednosti signalu.

```
library ieee;
use ieee.std_logic_1164.all;

entity koder4na2 is
        port (x0:       in std_logic;   -- ulazni port podataka 0
              x1:       in std_logic;   -- ulazni port podataka 1
              x2:       in std_logic;   -- ulazni port podataka 2
              x3:       in std_logic;   -- ulazni port podataka 3
              y0:       out std_logic;  -- izlazni port podataka 0
              y1:       out std_logic);-- izlazni port podataka 1
```

```
end entity koder4na2;

architecture beh1 of koder4na2 is
begin
        y0 <= '1' when x3 = '1' and x2 = '0' and
                        x1 = '0' and x0 = '0' else
              '1' when x3 = '0' and x2 = '0' and
                        x1 = '1' and x0 = '0' else
              '0';

        y1 <= '1' when x3 = '1' and x2 = '0' and
                        x1 = '0' and x0 = '0' else
              '1' when x3 = '0' and x2 = '1' and
                        x1 = '0' and x0 = '0' else
              '0';
end architecture beh1;
```

Verifikaciono okruženje

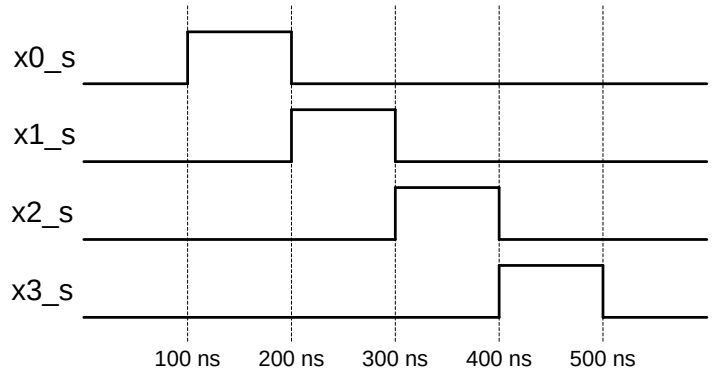
Primer jednog mogućeg testbenča koji se može iskoristiti za verifikaciju binarnog kodera 4-na-2 prikazan je u nastavku.

```
library ieee;
use ieee.std_logic_1164.all;

entity koder4na2_tb is
end entity koder4na2_tb;

architecture beh of koder4na2_tb is
   -- Deklaracija komponente VHDL modula koji se verifikuje (DUV)
   -- U ovom slučaju je to binarni koder 4-na-2
   component koder4na2 is
        port (x0:    in std_logic;   -- ulazni port podataka 0
              x1:    in std_logic;   -- ulazni port podataka 1
              x2:    in std_logic;   -- ulazni port podataka 2
              x3:    in std_logic;   -- ulazni port podataka 3
              y0:    out std_logic;  -- izlazni port podataka 0
              y1:    out std_logic);-- izlazni port podataka 1
   end component koder4na2;

   -- Deklaracija unutrašnjih signala potrebnih za povezivanje stimulus
   -- generatora sa ulazima DUV-a
   signal x0_s, x1_s, x2_s, x3_s: std_logic;
   signal y0_s, y1_s: std_logic;

begin
        -- Komponenta koja se verifikuje
        duv: koder4na2
           port map (
              x0 => x0_s,
              x1 => x1_s,
```

```
        x2 => x2_s,
        x3 => x3_s,
        y0 => y0_s,
        y1 => y1_s);

    -- Stimulus generator koji generise potrebne vrednosti na
    -- ulaznim portovima DUV-a na osnovu kojih ce biti moguce
    -- proveriti da li DUV implementira potrebnu funkcionalnost
    stim_gen: process
    begin
            x0_s <= '0', '1' after 100 ns, '0' after 200 ns;

            x1_s <= '0', '1' after 200 ns, '0' after 300 ns;

            x2_s <= '0', '1' after 300 ns, '0' after 400 ns;

            x3_s <= '0', '1' after 400 ns, '0' after 500 ns;

            wait;
    end process;
end architecture beh;
```

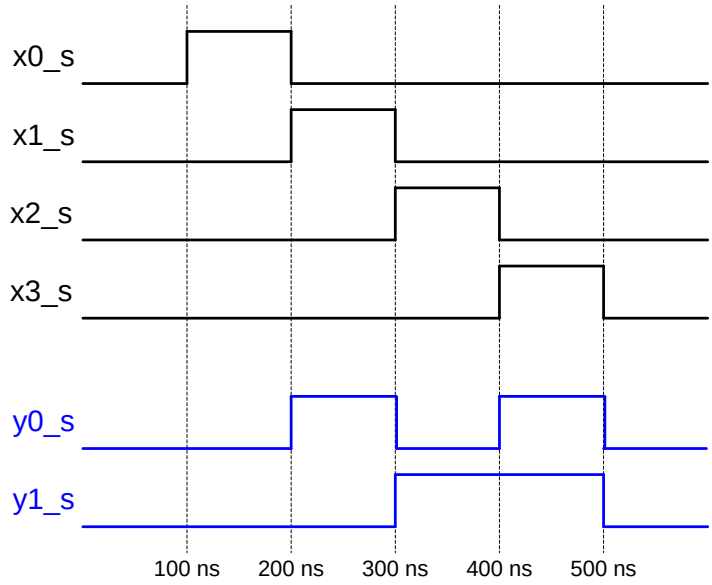
Prikazani testbenč instancionira komponentu binarnog kodera 4-na-2 i koristeći *stim_gen* proces generiše povorku ulaznih signala koja može da se iskoristi za proveru ispravnog rada napisanog modela binarnog kodera.

Stim_gen proces generiše vremenske oblike za ukupno četiri signala, *x0_s*, *x1_s*, *x2_s* i *x3_s*, koristeći četiri sekvencijalne naredbe dodele vrednosti signalu. Talasni oblici koji će biti generisani pomoću ove četiri naredbe dodele vrednosti signalu prikazani su na slici 2.



Slika 2. Generisani talasni oblici na četiri ulazna signala binarnog kodera 4-na-2

Nakon što se izvrši simulacija kombinovanog rada testbenča i modela binarnog kodera talasni oblik izlaznih signala podataka binarnog kodera *y0_s* i *y1_s*, trebalo bi da izgleda kao na slici 3.



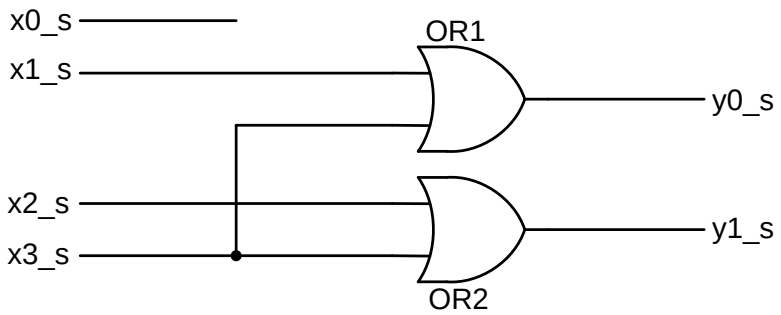
Slika 3. Generisani talasni oblici na četiri ulazna signala binarnog kodera 4-na-2 zajedno sa talasnim oblicima izlaznih signala podataka *y0_s* i *y1_s* koji su dobijeni kao rezultat simulacije

NAPOMENA: Razvijeno verifikaciono okruženje je univerzalno i može se koristiti za funkcionalnu verifikaciju bilo kojeg od predloženih modela binarnog kodera 4-na-2.

Zadaci za vežbu

Zadatak 1:

Napisati strukturni VHDL model binarnog kodera 4-na-2 baziran na šematskom prikazu sa slike 4. Napisati i sve potrebne modele standardnih logičkih kapija koje se koriste u modelu. Izvršiti funkcionalnu verifikaciju razvijenog modela.



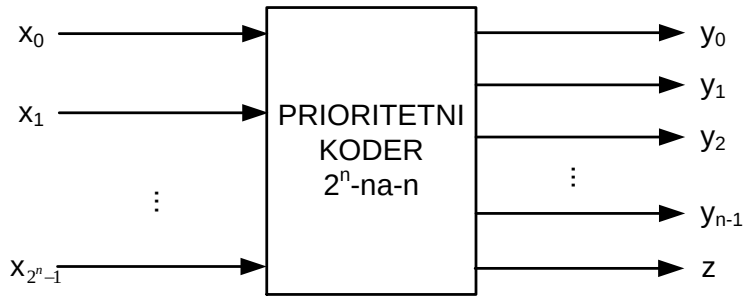
Slika 4. Struktni model binarnog kodera 4-na-2

Zadatak 2:

Napisati bihevijanlni VHDL model binarnog kodera 8-na-3. Za razvijeni VHDL model napisati testbenč koji se može koristiti za funkcionalnu verifikaciju.

Prioritetni koder

Kod prioritetnog kodera svaki od ulaza ima prioritet koji mu je pridružen. Izlazi prioritetnog kodera predstavljaju indikaciju aktivnog ulaza sa najvećim prioritetom. Kada je ulaz sa najvišim prioritetom aktivan, svi ulazi sa nižim prioritetima se ignorišu. Standardni interfejs prioritetnog kodera prikazan je na slici 5 i sastoji se iz 2^n ulaznih portova, širine 1 bit, *n* izlaznih portova, takođe širine 1 bit i jednog dodatnog izlaznog porta, *z*, širine 1 bit.



Slika 5. Interfejs generičkog prioritetnog kodera 2^n-na-*n*

Kao ilustracija rada prioritetnog kodera tablica istinitosnih dodela prioritetnog kodera 4-na-2 prikazana je dole.

x_3	x_2	x_1	x_0	y_1	y_0	z
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

U gornjoj tabeli pretpostavljeno je da je prioritet ulaznog porta x_0 najniži, a prioritet ulaznog porta x_3 najviši. Izlazni portovi y_0 i y_1 predstavljaju binarni broj koji identifikuje trenutno aktivni ulazni port sa najvišim prioritetom. Obzirom da postoji mogućnost da ni jedan od ulaznih portova nije aktivan, izlazni port *z* služi za signalizaciju ovog slučaja. U slučaju da su svi ulazni portovi neaktivni, izlazni port *z* ima vrednost 0. U ovom slučaju izlazi y_0 i y_1 nemaju smisao, tako da se njihove vrednosti ignorišu, odnosno mogu imati proizvoljne vrednosti. U slučaju da je barem jedan ulazni port aktivan, izlazni port *z* ima vrednost 1.

VHDL modeli prioritetnog kodera

Da bi smo ilustrovali način modelovanja prioritetni koder 4-na-2 čija je tablica istinitosnih dodela prikazana ranije. Portovi prioritetnog kodera 4-na-2 su:

- četiri1-bitna ulazna porta za podatke, x_0, x_1, x_2 i x_3
- dva 1-bitna izlaza za podatke, y_0 i y_1
- jedan 1-bitni izlaz za indikaciju situacija kada ni jedan ulaz nije aktivan, *z*

Odogovarajuća *entity* deklaracija prioritetnog kodera 4-na-2 ima sledeći izgled

```
library ieee;
use ieee.std_logic_1164.all;

entity prioritetni_koder4na2 is
        port (x0:       in std_logic;   -- ulazni port podataka 0
              x1:       in std_logic;   -- ulazni port podataka 1
              x2:       in std_logic;   -- ulazni port podataka 2
              x3:       in std_logic;   -- ulazni port podataka 3
              y0:       out std_logic;  -- izlazni port podataka 0
              y1:       out std_logic;  -- izlazni port podataka 1
              z:        out std_logic);-- indikator neaktivnosti
end entity prioritetni_koder4na2;
```

Nakon što smo opisali interfejs prioritetnog kodera, potrebno je modelovati njegovu funkcionalnost. Za ovo je potrebno napisati odgovarajuće arhitekturno telo.

Bihevijani modeli

Model prioritetnog kodera baziran na korišćenju *process* i *if* naredbe

```
architecture beh1 of prioritetni_koder4na2 is
begin
        koder: process (x0, x1, x2, x3) is
        begin
                if (x3 = '1') then
                        y1 <= '1';
                        y0 <= '1';
                elsif (x2 = '1') then
                        y1 <= '1';
                        y0 <= '0';
                elsif (x1 = '1') then
                        y1 <= '0';
                        y0 <= '1';
                else
                        y1 <= '0';
                        y0 <= '0';
                end if;

                if (x3 = ‘0’ and x2 = ‘0’ and
                   x1 = ‘0’ and x0 = ‘0’) then
                        z <= ‘0’;
                else
                        z <= ‘1’;
                end if;
        end process;
end architecture beh1;
```

Kombinovanjem entity deklaracije i odgovarajućeg arhitekturnog tela dobijamo kompletan model prioritetnog kodera 4-na-2. Na primer, dole je prikazan model

prioritetnog kodera 4-na-2 koji se bazira na funkcionalnom modelu koji koristi *process* naredbu.

```
library ieee;
use ieee.std_logic_1164.all;

entity prioritetni_koder4na2 is
        port (x0:       in std_logic;   -- ulazni port podataka 0
              x1:       in std_logic;   -- ulazni port podataka 1
              x2:       in std_logic;   -- ulazni port podataka 2
              x3:       in std_logic;   -- ulazni port podataka 3
              y0:       out std_logic;  -- izlazni port podataka 0
              y1:       out std_logic;  -- izlazni port podataka 1
              z:        out std_logic);-- indikator neaktivnosti
end entity prioritetni_koder4na2;


architecture beh1 of prioritetni_koder4na2 is
begin
        koder: process (x0, x1, x2, x3) is
        begin
                if (x3 = '1') then
                        y1 <= '1';
                        y0 <= '1';
                elsif (x2 = '1') then
                        y1 <= '1';
                        y0 <= '0';
                elsif (x1 = '1') then
                        y1 <= '0';
                        y0 <= '1';
                else
                        y1 <= '0';
                        y0 <= '0';
                end if;

                if (x3 = ‘0’ and x2 = ‘0’ and
                   x1 = ‘0’ and x0 = ‘0’) then
                        z <= ‘0’;
                else
                        z <= ‘1’;
                end if;
        end process;
end architecture beh1;
```

Verifikaciono okruženje

Primer jednog mogućeg testbenča koji se može iskoristiti za verifikaciju prioritetnog kodera 4-na-2 prikazan je u nastavku.

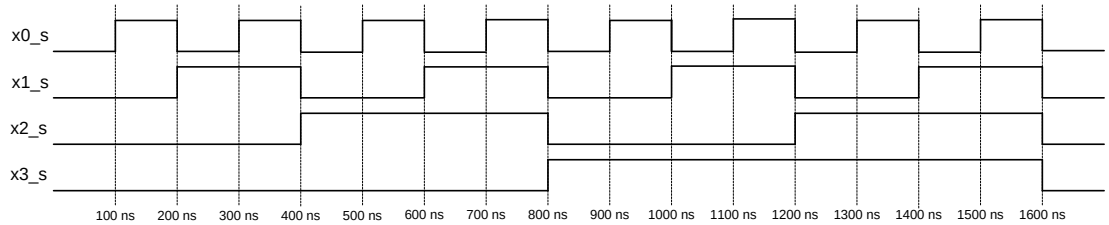
```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity prioritetni_koder4na2_tb is
end entity prioritetni_koder4na2_tb;


architecture beh of prioritetni_koder4na2_tb is
   -- Deklaracija komponente VHDL modula koji se verifikuje (DUV)
   -- U ovom slučaju je to prioritetni koder 4-na-2
   component prioritetni_koder4na2 is
         port (x0:      in std_logic;   -- ulazni port podataka 0
               x1:      in std_logic;   -- ulazni port podataka 1
               x2:      in std_logic;   -- ulazni port podataka 2
               x3:      in std_logic;   -- ulazni port podataka 3
               y0:      out std_logic;  -- izlazni port podataka 0
               y1:      out std_logic;  -- izlazni port podataka 1
               z:       out std_logic);-- indikator aktivnosti
   end component prioritetni_koder4na2;

   -- Deklaracija unutrašnjih signala potrebnih za povezivanje stimulus
   -- generatora sa ulazima DUV-a
   signal x0_s, x1_s, x2_s, x3_s: std_logic;
   signal y0_s, y1_s: std_logic;
   signal z_s: std_logic;

begin
         -- Komponenta koja se verifikuje
         duv: prioritetni_koder4na2
            port map (
               x0 => x0_s,
               x1 => x1_s,
               x2 => x2_s,
               x3 => x3_s,
               y0 => y0_s,
               y1 => y1_s,
               z => z_s);

         -- Stimulus generator koji generise potrebne vrednosti na
         -- ulaznim portovima DUV-a na osnovu kojih ce biti moguce
         -- proveriti da li DUV implementira potrebnu funkcionalnost
         stim_gen: process
         begin
               x0_s <= '0', '1' after 100 ns, '0' after 200 ns,
                           '1' after 300 ns, '0' after 400 ns,
                           '1' after 500 ns, '0' after 600 ns,
                           '1' after 700 ns, '0' after 800 ns,
                           '1' after 900 ns, '0' after 1000 ns,
                           '1' after 1100 ns, '0' after 1200 ns,
                           '1' after 1300 ns, '0' after 1400 ns,
                           '1' after 1500 ns, '0' after 1600 ns;

               x1_s <= '0', '1' after 200 ns, '0' after 400 ns,
                           '1' after 600 ns, '0' after 800 ns,
                           '1' after 1000 ns, '0' after 1200 ns,
```

```
                    '1' after 1400 ns, '0' after 1600 ns;

            x2_s <= '0', '1' after 400 ns, '0' after 800 ns,
                    '1' after 1200 ns, '0' after 1600 ns;

            x3_s <= '0', '1' after 800 ns, '0' after 1600 ns;

            wait;
        end process;
    end architecture beh;
```

Prikazani testbenč instancionira komponentu prioritetnog kodera 4-na-2 i koristeći *stim_gen* proces generiše povorku ulaznih signala koja može da se iskoristi za proveru ispravnog rada napisanog modela prioritetnog kodera.

Stim_gen proces generiše vremenske oblike za ukupno četiri signala, *x0_s*, *x1_s*, *x2_s* i *x3_s*, koristeći četiri sekvencijalne naredbe dodele vrednosti signalu. Talasni oblici koji će biti generisani pomoću ove četiri naredbe dodele vrednosti signalu prikazani su na slici 6.

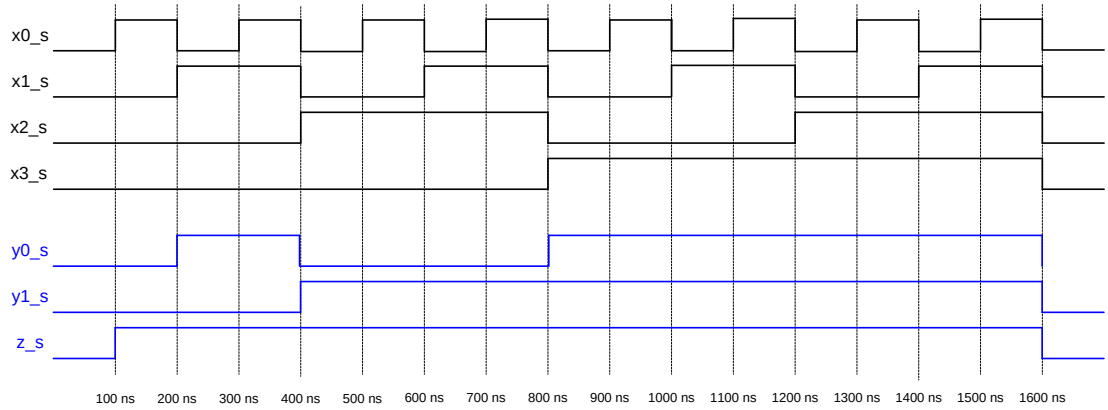


Slika 6. Generisani talasni oblici na četiri ulazna signala prioritetnog kodera 4-na-2

```
stim_gen: process
        subtype small_int is integer range 0 to 15;
        variable counter_v: small_int := 0;
    begin
        for i in 1 to 17 loop
            x0_s <= conv_std_logic_vector(counter_v, 4)(0);
            x1_s <= conv_std_logic_vector(counter_v, 4)(1);
            x2_s <= conv_std_logic_vector(counter_v, 4)(2);
            x3_s <= conv_std_logic_vector(counter_v, 4)(3);
            counter_v := counter_v + 1;
            wait for 100 ns;
        end loop;
        wait;
    end process;
```

NAPOMENA: Ukoliko se koristi gore prikazani *stim_gen* proces, u testbenč modulu neophodno je uključiti i *std_logic_arith* paket iz *ieee* biblioteke.

Nakon što se izvrši simulacija kombinovanog rada testbenča i modela prioritetnog talasni oblik izlaznog signala podataka prioritetnog *y0_s*, *y1_s* i *z_s*, trebalo bi da izgleda kao na slici 7.



Slika 7. Generisani talasni oblici na četiri ulazna signala prioritetnog kodera 4-na-2 zajedno sa talasnim oblicima izlaznih signala podataka *y0_s* i *y1_s* i signala indikacije aktivnosti *z*, koji su dobijeni kao rezultat simulacije

NAPOMENA: Razvijeno verifikaciono okruženje je univerzalno i može se koristiti za funkcionalnu verifikaciju bilo kojeg od predloženih modela prioritetnog kodera 4-na-2.

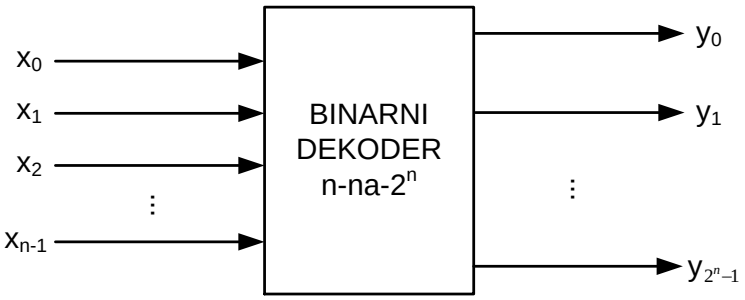
Zadaci za vežbu

Zadatak 1:

Napisati bihevijalni VHDL model prioritetnog kodera 8-na-3. Za razvijeni VHDL model napisati testbenč koji se može koristiti za funkcionalnu verifikaciju.

Dekoder

Dekoder je kombinaciona mreža koja služi za dekodovanje odgovarajuće kodovane informacije. U opštem slučaju binarni dekoder, prikazan na slici 8, ima *n* ulaznih i 2^n izlaznih portova. Svi ulazni i izlazni portovi su širine 1 bit.



Slika 8. Interfejs generičkog binarnog dekodera

U normalnom režimu rada binarnog dekodera za svaku moguću kombinaciju ulaznih signala tačno jedan od izlaza ima vrednost 1. Na primer, tablica istinitosnih dodela binarnog dekodera 2-na-4 prikazana je dole.

x_1	x_0	y_3	y_2	y_1	y_0
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1
x	x	0	0	0	0

VHDL modeli binarnog kodera

U našem primeru binarni dekoder 2-na-4 ima sledeće portove:

- dva 1-bitna ulaza za podatke, x_0 i x_1
- četiri1-bitna izlazna porta za podatke, y_0, y_1, y_2 i y_3

Odogovarajuća *entity* deklaracija binarnog dekodera 2-na-4 ima sledeći izgled

```
library ieee;
use ieee.std_logic_1164.all;

entity dekoder2na4 is
        port (x0:    in std_logic;   -- ulazni port podataka 0
              x1:    in std_logic;   -- ulazni port podataka 1
              y0:    out std_logic;  -- izlazni port podataka 0
              y1:    out std_logic;  -- izlazni port podataka 1
              y2:    out std_logic;  -- izlazni port podataka 2
              y3:    out std_logic); -- izlazni port podataka 3
end entity dekoder2na4;
```

Nakon što smo opisali interfejs binarnog kodera, potrebno je modelovati njegovu funkcionalnost. Za ovo je potrebno napisati odgovarajuće arhitekturno telo. Obzirom

da se binarni koder može modelovati na razne načine, možemo napisati veliki broj različitih arhitekturnih tela.

NAPOMENA: U narednim arhitekturnim telima pretpostavka je da su svi ulazni i izlazni portovi tipa *std_logic*.

Bihevijani modeli

Varijanta 1: Model binarnog dekodera baziran na korišćenju konkurentne naredbe uslovne dodele vrednosti signalu

```
architecture beh1 of dekoder2na4 is
begin
        y0 <= ‘1’ when  x1 = ‘0’ and x0 = ‘0’ else
                ‘0’;
        y1 <= ‘1’ when  x1 = ‘0’ and x0 = ‘1’ else
                ‘0’;
        y2 <= ‘1’ when  x1 = ‘1’ and x0 = ‘0’ else
                ‘0’;
        y3 <= ‘1’ when  x1 = ‘1’ and x0 = ‘1’ else
                ‘0’;
end architecture beh1;
```

Varijanta 2: Model binarnog dekodera baziran na korišćenju *process* i *if* naredbe

```
architecture beh2 of dekoder2na4 is
begin
        dekoder: process (x0, x1, x2, x3) is
        begin
                if (x1 = '0' and x0 = '0') then
                        y0 <= '1';
                else
                        y0 <= '0';
                end if;
                if (x1 = '0' and x0 = '1') then
                        y1 <= '1';
                else
                        y1 <= '0';
                end if;
                if (x1 = '1' and x0 = '0') then
                        y2 <= '1';
                else
                        y2 <= '0';
                end if;
                if (x1 = '1' and x0 = '1') then
                        y3 <= '1';
                else
                        y3 <= '0';
                end if;
```

```
        end process;
end architecture beh2;
```

Verifikaciono okruženje

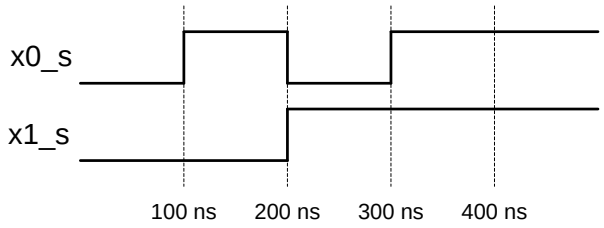
Primer jednog mogućeg testbenča koji se može iskoristiti za verifikaciju binarnog dekodera 2-na-4 prikazan je u nastavku.

```
library ieee;
use ieee.std_logic_1164.all;

entity dekoder2na4_tb is
end entity dekoder2na4_tb;


architecture beh of dekoder2na4_tb is
   -- Deklaracija komponente VHDL modula koji se verifikuje (DUV)
   -- U ovom slučaju je to binarni koder 4-na-2
   component dekoder2na4is
        port (x0:       in std_logic;    -- ulazni port podataka 0
              x1:       in std_logic;    -- ulazni port podataka 1
              y0:       out std_logic;  -- izlazni port podataka 0
              y1:       out std_logic;  -- izlazni port podataka 1
              y2:       out std_logic;  -- izlazni port podataka 2
              y3:       out std_logic); -- izlazni port podataka 3
   end component dekoder2na4;

   -- Deklaracija unutrašnjih signala potrebnih za povezivanje stimulus
   -- generatora sa ulazima DUV-a
   signal x0_s, x1_s: std_logic;
   signal y0_s, y1_s, y2_s, y3_s: std_logic;

begin
        -- Komponenta koja se verifikuje
        duv: dekoder2na4
           port map (
              x0 => x0_s,
              x1 => x1_s,
              y0 => y0_s,
              y1 => y1_s,
              y2 => y2_s,
              y3 => y3_s);

        -- Stimulus generator koji generise potrebne vrednosti na
        -- ulaznim portovima DUV-a na osnovu kojih ce biti moguce
        -- proveriti da li DUV implementira potrebnu funkcionalnost
        stim_gen: process
        begin
                x0_s <= '0', '1' after 100 ns, '0' after 200 ns,
                                '1' after 300 ns;
```

```
        x1_s <= '0', '1' after 200 ns;

        wait;
    end process;
end architecture beh;
```

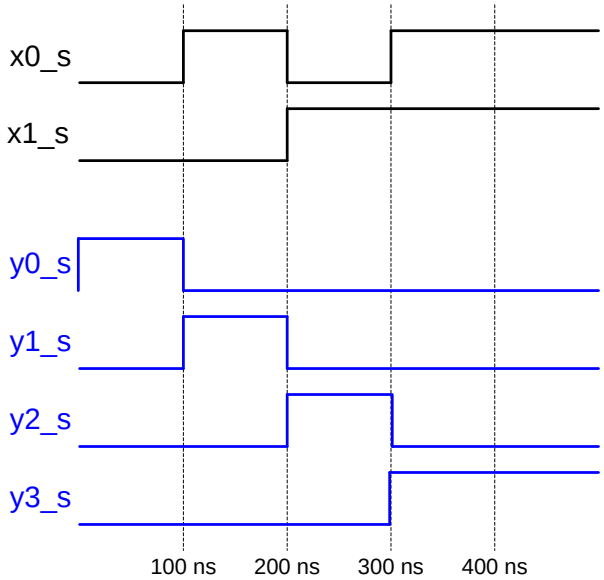
Prikazani testbenč instancionira komponentu binarnog dekodera 2-na-4 i koristeći *stim_gen* proces generiše povorku ulaznih signala koja može da se iskoristi za proveru ispravnog rada napisanog modela binarnog dekodera.

Stim_gen proces generiše vremenske oblike za ukupno dva signala, *x0_s*, *x1_s*, koristeći dve sekvencijalne naredbe dodele vrednosti signalu. Talasni oblici koji će biti generisani pomoću ove dve naredbe dodele vrednosti signalu prikazani su na slici 9.



Slika 9. Generisani talasni oblici na četiri ulazna signala binarnog dekodera 2-na-4

Nakon što se izvrši simulacija kombinovanog rada testbenča i modela binarnog kodera talasni oblik izlaznih signala podataka binarnog dekodera *y0_s*, *y1_s*, *y2_s* i *y3_s*, trebalo bi da izgleda kao na slici 10.



Slika 10. Generisani talasni oblici na četiri ulazna signala binarnog dekodera 2-na-4 zajedno sa talasnim oblicima izlaznih signala podataka *y0_s*, *y1_s*, *y2_s* i *y3_s* koji su dobijeni kao rezultat simulacije

NAPOMENA: Razvijeno verifikaciono okruženje je univerzalno i može se koristiti za funkcionalnu verifikaciju bilo kojeg od predloženih modela binarnog dekodera 2-na-4.

Zadaci za vežbu

Zadatak 1:

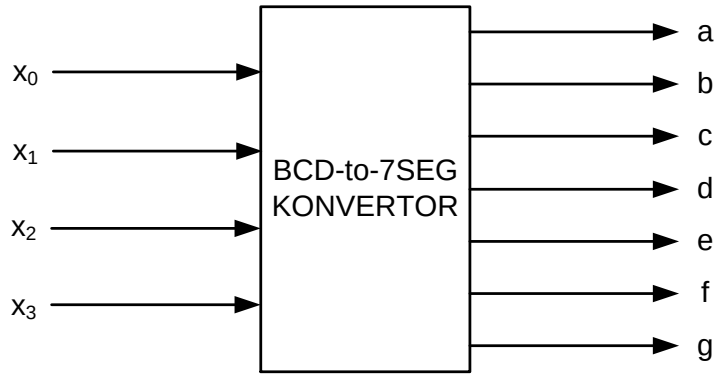
Preraditi prikazani model binarnog dekodera 2-na-4 tako da uključuje dodatni port dozvole rada, *en*. Kada je *en* port postavljen na 0, dekoder treba da je u neaktivnom stanju u kome se na njegovim izlazima stalno drži vrednost "0000", bez obzira na promene i vrednosti ulaznih portova. Kada je en port postavljen na 1, dekoder je aktivan i treba da vrši svoju funkciju. Napisati testbenč koji se može koristiti za funkcionalnu verifikaciju razvijenog VHDL modela binarnog dekodera sa dozvolom rada.

Zadatak 2:

Napisati bihevijanlni VHDL model binarnog dekodera 3-na-8. Za razvijeni VHDL model napisati testbenč koji se može koristiti za funkcionalnu verifikaciju.

Konvertori koda

Svrha konvertora koda je da izvrše konverziju iz jednog tipa kodovanja u neki drugi. Postoji veliki broj različitih konvertora koda, od kojih je najpoznatiji konvetor BCD (*binary coded decimal*, binarno kodirana decimalna cifra) koda u 7-segmentni kod koji je pogodan za kontrolisanje 7-segmentnog displeja. Interfejs ovog konvertora koda prikazan je na slici 11.



Slika 11. Interfejs BCD-to-7SEG konvetora koda

Tablica istinitosnih dodela BCD-to-7seg konvetora koda prikazana je u nastavku.

x_3	x_2	x_1	x_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

VHDL model BCD-to-7SEG konvertora

Portovi BCD-to-7SEG konvertora su:

- četiri1-bitna ulazna porta za podatke, x_0, x_1, x_2 i x_3
- 7 1-bitna izlaza za podatke, *a, b, c, d, e, f, g*

Odogovarajuća *entity* deklaracija BCD-to-7SEG konvertora ima sledeći izgled

```
library ieee;
use ieee.std_logic_1164.all;

entity bcd_to_7seg is
        port (x0:       in std_logic;   -- ulazni port podataka 0
              x1:       in std_logic;   -- ulazni port podataka 1
              x2:       in std_logic;   -- ulazni port podataka 2
              x3:       in std_logic;   -- ulazni port podataka 3
              a:        out std_logic;  -- a segment
```

```
        b:      out std_logic;  -- b segment
        c:      out std_logic;  -- c segment
        d:      out std_logic;  -- d segment
        e:      out std_logic;  -- e segment
        f:      out std_logic;  -- f segment
        g:      out std_logic);-- g segment
end entity bcd_to_7seg;
```

Nakon što smo opisali interfejs prioritetnog kodera, potrebno je modelovati njegovu funkcionalnost. Za ovo je potrebno napisati odgovarajuće arhitekturno telo.

Bihevijani model

Model BCD-to-7SEG konvertora baziran na korišćenju *process* i *if* naredbe

```
architecture beh of bcd_to_7seg is
begin
    konvertor: process (x0, x1, x2, x3) is
      variable input_vector_v: std_logic_vector(3 downto 0);
    begin
      input_vector_v := x3&x2&x1&x0;
      case input_vector_v is
        when “0000” =>
          a <= ‘1’;
          b <= ‘1’;
          c <= ‘1’;
          d <= ‘1’;
          e <= ‘1’;
          f <= ‘1’;
          g <= ‘0’;
        when “0001” =>
          a <= ‘0’;
          b <= ‘1’;
          c <= ‘1’;
          d <= ‘0’;
          e <= ‘0’;
          f <= ‘0’;
          g <= ‘0’;
        when “0010” =>
          a <= ‘1’;
          b <= ‘1’;
          c <= ‘0’;
          d <= ‘1’;
          e <= ‘1’;
          f <= ‘0’;
          g <= ‘1’;
        when “0011” =>
          a <= ‘1’;
          b <= ‘1’;
          c <= ‘1’;
          d <= ‘1’;
```

```
            e <= '0';
            f <= '0';
            g <= '1';
        when "0100" =>
            a <= '0';
            b <= '1';
            c <= '1';
            d <= '0';
            e <= '0';
            f <= '1';
            g <= '1';
        when "0101" =>
            a <= '1';
            b <= '0';
            c <= '1';
            d <= '1';
            e <= '0';
            f <= '1';
            g <= '1';
        when "0110" =>
            a <= '1';
            b <= '0';
            c <= '1';
            d <= '1';
            e <= '1';
            f <= '1';
            g <= '1';
        when "0111" =>
            a <= '1';
            b <= '1';
            c <= '1';
            d <= '0';
            e <= '0';
            f <= '0';
            g <= '0';
        when "1000" =>
            a <= '1';
            b <= '1';
            c <= '1';
            d <= '1';
            e <= '1';
            f <= '1';
            g <= '1';
        when others =>
            a <= '1';
            b <= '1';
            c <= '1';
            d <= '1';
            e <= '0';
            f <= '1';
            g <= '1';
    end case;
end process;
```

```
end architecture beh;
```

Verifikaciono okruženje

Primer jednog mogućeg testbenča koji se može iskoristiti za verifikaciju BCD-to-7SEG konvertora prikazan je u nastavku.

```
library ieee;
use ieee.std_logic_1164.all;

entity bcd_to_7seg _tb is
end entity bcd_to_7seg _tb;


architecture beh of bcd_to_7seg _tb is
   -- Deklaracija komponente VHDL modula koji se verifikuje (DUV)
   -- U ovom slučaju je to prioritetni koder 4-na-2
   component bcd_to_7seg is
         port (x0:      in std_logic;   -- ulazni port podataka 0
               x1:      in std_logic;   -- ulazni port podataka 1
               x2:      in std_logic;   -- ulazni port podataka 2
               x3:      in std_logic;   -- ulazni port podataka 3
               a:       out std_logic;  -- a segment
               b:       out std_logic;  -- b segment
               c:       out std_logic;  -- c segment
               d:       out std_logic;  -- d segment
               e:       out std_logic;  -- e segment
               f:       out std_logic;  -- f segment
               g:       out std_logic);-- g segment
   end component bcd_to_7seg;

   -- Deklaracija unutrašnjih signala potrebnih za povezivanje stimulus
   -- generatora sa ulazima DUV-a
   signal x0_s, x1_s, x2_s, x3_s: std_logic;
   signal a_s, b_s, c_s, d_s, e_s, f_s, g_s: std_logic;

begin
        -- Komponenta koja se verifikuje
        duv: bcd_to_7seg
           port map (
              x0 => x0_s,
              x1 => x1_s,
              x2 => x2_s,
              x3 => x3_s,
              a => a_s,
              b => b_s,
              c => c_s,
              d => d_s,
              e => e_s,
              f => f_s,
              g => g_s);
```

```
    -- Stimulus generator koji generise potrebne vrednosti na
    -- ulaznim portovima DUV-a na osnovu kojih ce biti moguce
    -- proveriti da li DUV implementira potrebnu funkcionalnost
    stim_gen: process
      subtype small_int is integer range 0 to 15;
      variable counter_v: small_int := 0;
    begin
      for i in 1 to 10 loop
        x0_s <= conv_std_logic_vector(counter_v, 4)(0);
        x1_s <= conv_std_logic_vector(counter_v, 4)(1);
        x2_s <= conv_std_logic_vector(counter_v, 4)(2);
        x3_s <= conv_std_logic_vector(counter_v, 4)(3);
        counter_v := counter_v + 1;
        wait for 100 ns;
      end loop;
      wait;
    end process;
end architecture beh;
```

Prikazani testbenč instancionira komponentu BCD-to-7SEG konvertora i koristeći *stim_gen* proces generiše povorku ulaznih signala koja može da se iskoristi za proveru ispravnog rada napisanog modela konvertora.

Zadaci za vežbu

Zadatak 1:

Prikazani VHDL model BCD-to-7SEG konvertora je previše složen. Ovo je prvenstveno zbog toga što se ulazni i izlazni podaci vode preko skalarnih portova. Model bi se mogao znatno pojednostaviti ukoliko bi se svi skalarni ulazni portovi objedinili u jedan vektorski port, *x*, koji bi bio tipa *std_logic_vector(3 downto 0)*. Slično, svi skalarni izlazni portovi mogli bi se objediniti u jedan vektorski, *disp*, koji bi bio tipa *std_logic_vector(6 downto 0)*. Napisati novi VHDL model BCD-to-7SEG konvertora koji bi bio baziran na korišćenju vektorskih portova ali koji bi i dalje koristio *process* naredbu. Napisati takođe i novi testbenč koji bi se mogao koristiti za verifikaciju.

Zadatak 2:

Pod pretpostavkama iz zadatka 1, napisati VHDL model BCD-to-7SEG konvertora sa vektorskim portovima, ali sada korišćenjem konkurentne selektovane naredbe dodele vrednosti signalu. Koristeći testbenč razvijen u zadatku 1, izvršiti verifikaciju napisanog modela.

Zadata 3:

Napisati VHDL model konvertora 3-bitnog binarnog koda u Grejev kod. Tablica istinitosnih dodela konvertora čiji je model neophodno razviti prikazana je u nastavku.

b_2	b_1	b_0	g_2	g_1	g_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

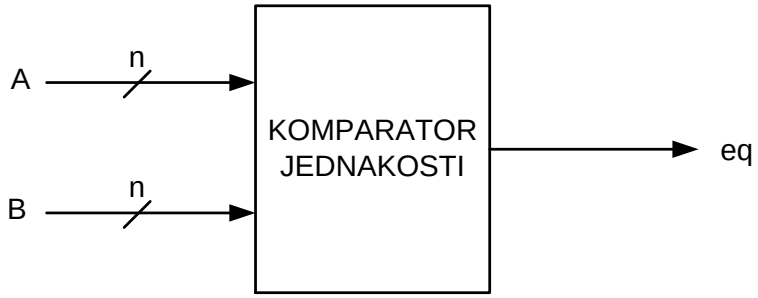
Za napisani VHDL model napisati i odgovarajući testbenč koji se može iskoristiti za funkcionalnu verifikaciju razvijenog modela.

Komparator

Komparator je kombinaciona mreža koja upoređuje relativne veličine dva binarna broja. Komparator obično ima dva ulazna porta, *A* i *B*, svaki širine *n* bita, i jedan ili više izlaznih portova od kojih je svaki širine 1 bit. Broj izlaznih portova zavisi od vrste komparatora.

Komparator jednakosti

U slučaju komparatora jednakosti (*identity comparator*), prikazanog na slici 12, postoji samo jedan izlazni port, obično označen sa *eq*, koji daje informaciju o tome da li su binarni brojevi koji su trenutno prisutni na ulaznim portovima jednaki ili nisu. U slučaju da su brojevi jednaki *eq* port ima vrednost 1, u protivnom ima vrednost 0.



Slika 12. N-bitni komparator jednakosti

VHDL modeli komparatora jednakosti

Da bi smo ilustovali VHDL modele komparatora jednakosti koristićemo 8-bitni komparator jednakosti koji ima sledeće portove:

- dva 8-bitna ulazna porta za podatke, *a* i *b*
- jedan1-bitni izlazni port za indikaciju jednakosti, *eq*

Odogovarajuća *entity* deklaracija 8-bitnog komparatora jednakosti ima sledeći izgled

```
library ieee;
use ieee.std_logic_1164.all;

entity komp_jednakosti is
   port (a: in std_logic_vector(7 downto 0);   -- ulazni port podataka 0
         b: in std_logic_vector(7 downto 0);   -- ulazni port podataka 1
         eq: out std_logic);                   -- izlazni port jednakosti
end entity komp_jednakosti;
```

Bihevijani modeli

Model komparatora jednakosti baziran na korišćenju *process* i *if* naredbe

```
architecture beh1 of komp_jednakosti is
begin
        komparator: process (a, b) is
        begin
                if (a = b) then
                        eq <= '1';
```

```
            else
                eq <= '0';
            end if;
        end process;
end architecture beh1;
```

Verifikaciono okruženje

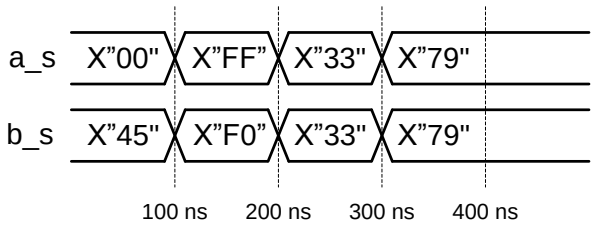
Primer jednog mogućeg testbenča koji se može iskoristiti za verifikaciju 8-bitnog komparatora jednakosti prikazan je u nastavku.

```
library ieee;
use ieee.std_logic_1164.all;

entity komp_jednakosti_tb is
end entity komp_jednakosti_tb;


architecture beh of komp_jednakosti_tb is
   -- Deklaracija komponente VHDL modula koji se verifikuje (DUV)
   -- U ovom slučaju je to komparator jednakosti
   component komp_jednakosti is
      port (a: in std_logic_vector(7 downto 0);-- ulazni port podataka 0
            b: in std_logic_vector(7 downto 0);-- ulazni port podataka 1
            eq: out std_logic);                -- izlazni port jednakosti
   end component komp_jednakosti;

   -- Deklaracija unutrašnjih signala potrebnih za povezivanje stimulus
   -- generatora sa ulazima DUV-a
   signal a_s, b_s: std_logic_vector(7 downto 0);
   signal eq_s: std_logic;

begin
        -- Komponenta koja se verifikuje
        duv: komp_jednakosti
           port map (
              a => a_s,
              b => b_s,
              eq => eq_s);

        -- Stimulus generator koji generise potrebne vrednosti na
        -- ulaznim portovima DUV-a na osnovu kojih ce biti moguce
        -- proveriti da li DUV implementira potrebnu funkcionalnost
        stim_gen: process
        begin
                a_s <= X“00”, X”FF” after 100 ns, X”33” after 200 ns,
                                X”79” after 300 ns;

                b_s <= X“45”, X”F0” after 100 ns, X”33” after 200 ns,
                                X”79” after 300 ns;

                wait;
```

```
    end process;
end architecture beh;
```

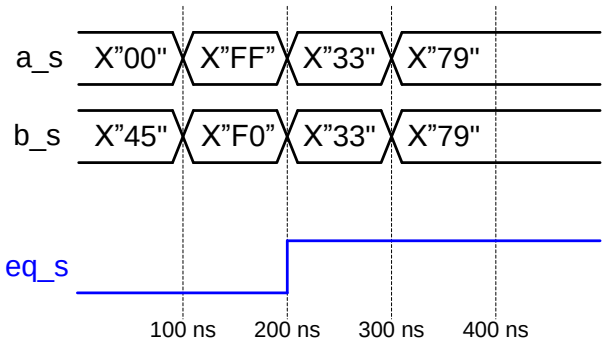
Prikazani testbenč instancionira komponentu 8-bitnog komparatora jednakosti i koristeći *stim_gen* proces generiše povorku ulaznih signala koja može da se iskoristi za proveru ispravnog rada napisanog modela komparatora jednakosti.

Stim_gen proces generiše vremenske oblike za ukupno dva signala, *a_s*, *b_s*, koristeći dve sekvencijalne naredbe dodele vrednosti signalu. Talasni oblici koji će biti generisani pomoću ove dve naredbe dodele vrednosti signalu prikazani su na slici 13.



Slika 13. Generisani talasni oblici na dva ulazna signala 8-bitnog komparatora jednakosti

Nakon što se izvrši simulacija kombinovanog rada testbenča i modela komparatora jednakosti talasni oblik izlaznog signala jednakosti *eq_s* trebalo bi da izgleda kao na slici 14.



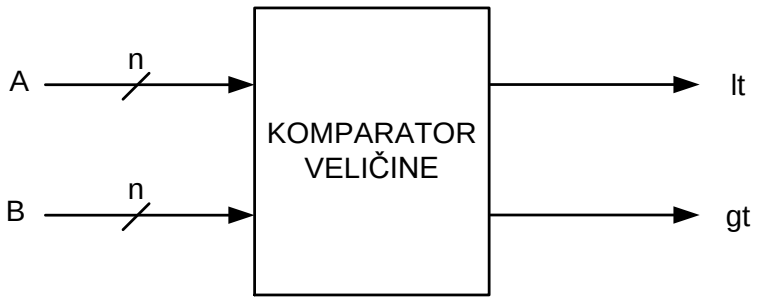
Slika 14. Generisani talasni oblici na dva ulazna signala 8-bitnog komparatora jednakosti zajedno sa talasnim oblikom izlaznog signala jednakosti *eq_s* koji je dobijen kao rezultat simulacije

Komparator veličine

U slučaju komparatora veličine (*magnitude comparator*), prikazanog na slici 15, postoji veći broj izlaznih portova, koji pružaju informaciju o odnosu binarnih brojeva koji su trenutno prisutni na ulaznim portovima. Izlazni portovi koji obično postoje su:

- port *lt*, koji označava da je broj a manji od broja b
- port *gt*, koji označava da je broj a veći od broja b
- port *lte*, koji označava da je broj a manji ili jednak sa brojem b
- port *gte*, koji označava da je broj a veći ili jednak sa brojem b

Na slici 15 prikazan je primer komparatora veličine koji ima samo dva od gore navedena četiri porta, *lt* i *gt*.



Slika 15. N-bitni komparator veličine sa *lt* i *gt* portovima

VHDL modeli komparatora jednakosti

Da bi smo ilustovali VHDL modele komparatora jednakosti koristićemo 8-bitni komparator veličine koji ima sledeće portove:

- dva 8-bitna ulazna porta za podatke, *a* i *b*
- 1-bitni izlazni port za indikaciju da je broj *a* manji od broja *b*, *lt*
- 1-bitni izlazni port za indikaciju da je broj *a* veći od broja *b*, *gt*

Odogovarajuća *entity* deklaracija 8-bitnog komparatora veličine ima sledeći izgled

```
library ieee;
use ieee.std_logic_1164.all;

entity komp_velicine is
   port (a: in std_logic_vector(7 downto 0);   -- ulazni port podataka 0
         b: in std_logic_vector(7 downto 0);   -- ulazni port podataka 1
         lt: out std_logic);                   -- izlazni port, broj a manji od b
         gt: out std_logic);                   -- izlazni port, broj a veci od b
end entity komp_velicine;
```

Bihevijani modeli

Model komparatora jednakosti baziran na korišćenju *process* i *if* naredbe

```
architecture beh1 of komp_velicine is
begin
        komparator: process (a, b) is
        begin
                if (a < b) then
                        lt <= '1';
                        gt <= '0';
                elsif (a > b) then
                        lt <= '0';
                        gt <= '1';
                else
                        lt <= '0';
                        gt <= '0';
                end if;
        end process;
end architecture beh1;
```

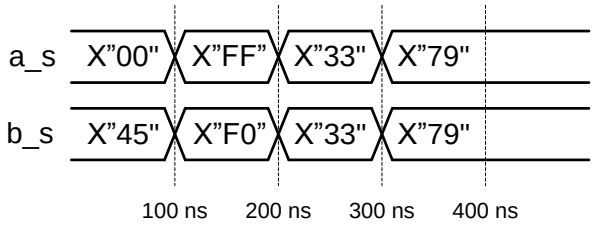
Verifikaciono okruženje

Primer jednog mogućeg testbenča koji se može iskoristiti za verifikaciju 8-bitnog komparatora veličine prikazan je u nastavku.

```
library ieee;
use ieee.std_logic_1164.all;

entity komp_velicine_tb is
end entity komp_velicine_tb;


architecture beh of komp_velicine_tb is
   -- Deklaracija komponente VHDL modula koji se verifikuje (DUV)
   -- U ovom slučaju je to komparator jednakosti
   component komp_velicine is
      port (a: in std_logic_vector(7 downto 0);-- ulazni port podataka 0
            b: in std_logic_vector(7 downto 0);-- ulazni port podataka 1
            lt: out std_logic);          -- izlazni port, broj a manji od b
            gt: out std_logic);          -- izlazni port, broj a veci od b
   end component komp_velicine;

   -- Deklaracija unutrašnjih signala potrebnih za povezivanje stimulus
   -- generatora sa ulazima DUV-a
   signal a_s, b_s: std_logic_vector(7 downto 0);
   signal lt_s, gt_s: std_logic;

begin
        -- Komponenta koja se verifikuje
        duv: komp_velicine
           port map (
              a => a_s,
              b => b_s,
              lt => lt_s,
              gt => gt_s);

        -- Stimulus generator koji generise potrebne vrednosti na
        -- ulaznim portovima DUV-a na osnovu kojih ce biti moguce
        -- proveriti da li DUV implementira potrebnu funkcionalnost
        stim_gen: process
        begin
                a_s <= X“00”, X”FF” after 100 ns, X”33” after 200 ns,
                                X”79” after 300 ns;

                b_s <= X“45”, X”F0” after 100 ns, X”33” after 200 ns,
                                X”79” after 300 ns;

                wait;
        end process;
end architecture beh;
```

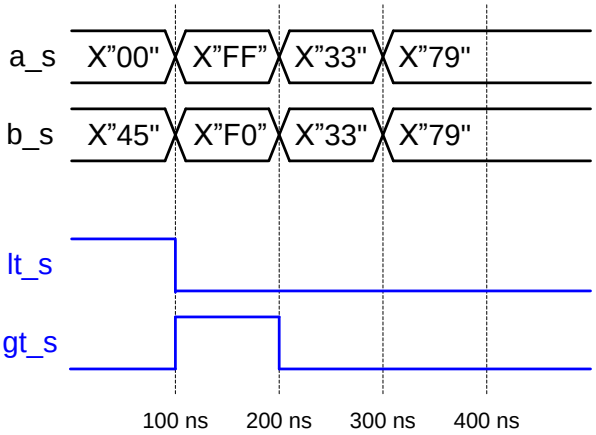
Prikazani testbenč instancionira komponentu 8-bitnog komparatora veličine i koristeći *stim_gen* proces generiše povorku ulaznih signala koja može da se iskoristi za proveru ispravnog rada napisanog modela komparatora veličine.

Stim_gen proces generiše vremenske oblike za ukupno dva signala, *a_s*, *b_s*, koristeći dve sekvencijalne naredbe dodele vrednosti signalu. Talasni oblici koji će biti generisani pomoću ove dve naredbe dodele vrednosti signalu prikazani su na slici 16.



Slika 16. Generisani talasni oblici na dva ulazna signala 8-bitnog komparatora veličine

Nakon što se izvrši simulacija kombinovanog rada testbenča i modela komparatora veličine talasni oblik izlaznih signala *lt_s* i *gt_s* trebalo bi da izgleda kao na slici 17.



Slika 17. Generisani talasni oblici na dva ulazna signala 8-bitnog komparatora veličine zajedno sa talasnim oblikcima izlaznih signala *lt_s* i *gt_s* koji su dobijeni kao rezultat simulacije

Zadaci za vežbu

Zadatak 1:

Napisati VHDL model koji realizuje 4-bitni komparator veličine koji ima tri izlazna porta, *eq*, *lt* i *gt*, koji daju informaciju o tome da li su ulazni 4-bitni brojevi jednaki, da li je broj *a* manji od broja *b* i da li je broj *a* veći od broja *b*. Napisati odgovarajući testbenč pomoću kojega je moguće izvršiti funkcionalnu verifikaciju razvijenog modela.

Zadatak 2:

Napisati VHDL model koji realizuje 8-bitni komparator veličine koji ima tri izlazna porta, *neq*, *lte* i *gte*, koji daju informaciju o tome da li su ulazni 8-bitni brojevi

različiti, da li je broj *a* manji ili jednak od broja *b* i da li je broj *a* veći ili jednak od broja *b*. Napisati odgovarajući testbenč pomoću kojega je moguće izvršiti funkcionalnu verifikaciju razvijenog modela.

Zadatak 3:

Napisati VHDL model koji realizuje komparator veličine koji ima jedan ulazni port *a*, i jedan izlazni port, *in_range*. Ukoliko je ulazni broj u opsegu $3 \leq a < 7$, izlazni port *in_range* treba da ima vrednost 1, u protivnom treba da ima vrednost 0. Napisati odgovarajući testbenč pomoću kojega je moguće izvršiti funkcionalnu verifikaciju razvijenog modela.